VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER ARCHITECTURE – CO2007

Assignment Report

4 IN A ROW

Advisor: Băng Ngọc Bảo Tâm

Student Name: Vũ Nam Bình

Student ID: 2152441

HO CHI MINH CITY, MARCH 2023

Contents

I) Introduction:

Four in a Row is a two-player connection game, in which each player chooses a game piece and takes turn to drop it into a column until there is one of them having 4 pieces that place continuously vertical, horizontal or diagonal. If after placing all empty position of the grid and no one has already connected 4 pieces, the game will end with a draw result.
In this assignment, I will develop the game mentioned below using MAR MIPS.

II) How to play:

Firstly, every player will be asked to choose their name, the first player's piece is 'X' while the second player's is 'O'.

Next, each player takes turn to make a move, respectively. In every turn, the player has 2 choices, which are "Drop a piece" or "Remove one arbitrary piece of their opponent". 2 players will play until there is a winner with 4 pieces placing continuously vertically, horizontaly or diagonally. If there are not any winners, the game will end with a draw result. Besides, there are also some function and also some rules that 2 players have to follow:
1) Each player has maximum 3 times to violate the rules below (if there exists one of two players violate more than 3 times, the other will immidiately be the winner).
2) In the first move, every player has to drop a piece in the center column. If you do not drop in the center, you will be counted 1 violation time and the turn is reset for you to choose again.

3) After your first move, each of you has 3 times to undo your own move (drop or remove). If you use this function more than 3 times, it's also counted as 1 violation time. (After choosing remove or drop, you will be asked to undo your move)

4) About remove choice, each of you has only 1 time to remove one arbitrary of your opponent. In case, the cell you choose is empty or contains your own piece, you will be counted 1 time violation and you also have to choose the cell to remove again so be careful with your choice! If you did remove one time before (not undo) but still try to remove 1 more time, you will be counted 1 time violation and your turn will be reset.

5) About drop choice, you have to drop into valid column (from column 1 to 7) In case you drop out of the board or you drop into a full column, this is counted as 1 time violation! Your turn will be reset.

Note:

1) If you choose to drop a piece, the game will ask you to choose a column. Now just type a column that you want and press "ENTER".

2) If you choose to remove your opponent's piece, the game will ask you to enter the position containing your opponent's piece that you want to remove. First, you have to type the row, press "ENTER" and then type the column, press "Enter".

3) With undo and block, the game ask you to choose 1 for Yes and 0 for No. If you type other numbers, it will ask again until you choose 1 or 0.

III) Algorithm:

Before explanation:

All the indexes for example board[i][j] that I'll use to explain are based on normal logic (I mean when I mention the cell having position row 3, column 4,

in reality it means board[3][4] if row and column starting from 1). However in the code I use this formula to calculate the address of the cell (because MAR MIPS is not the same as C, C++,… I can not use board[3][4] to access this address):

Address = Base + (Row Index * Column Size + Column Index)*dataSize

1) Base is the address of the board I declared in .data part.

2) Row Index is the index of the row (I will represent the formula to calculate this index later).

3) Column Size is the maximum column of the board (I declared a variable by using this .eqv colSize 29)

4) Column Index is the index of the row (I will represent the formula to calculate this index later).

5) DataSize is 1 (because the data is character 1 byte).

Formula to calculate Row Index: Let says A is the row input from player

=> After receiving input from player, my program change its value by using this formula: **A = 1 + (A - 1)*2**.

Formula to calculate Row Index: Let says B is the row input from player

=> After receiving input from player, my program change its value by using this formula: **B = 2 + (B - 1)*4**.

I changed the row and column because the board I used in this assignment including 29 columns and 13 rows and only columns 2, 6, 10, 14, 18, 22, 26 and rows 1, 3, 5, 7, 9, 11 are used to store pieces “X” or “O” so when I receive input from player, I have to changes its value.

1) About the board:

I declared a board with .ascii type in data part:

```
board: .ascii"+---+---+---+---+---+---+---+"
       .ascii"|   |   |   |   |   |   |   |"
       .ascii"+---+---+---+---+---+---+---+"
       .ascii"|   |   |   |   |   |   |   |"
       .ascii"+---+---+---+---+---+---+---+"
       .ascii"|   |   |   |   |   |   |   |"
       .ascii"+---+---+---+---+---+---+---+"
       .ascii"|   |   |   |   |   |   |   |"
       .ascii"+---+---+---+---+---+---+---+"
       .ascii"|   |   |   |   |   |   |   |"
       .ascii"+---+---+---+---+---+---+---+"
       .ascii"|   |   |   |   |   |   |   |"
       .ascii"+---+---+---+---+---+---+---+"
```

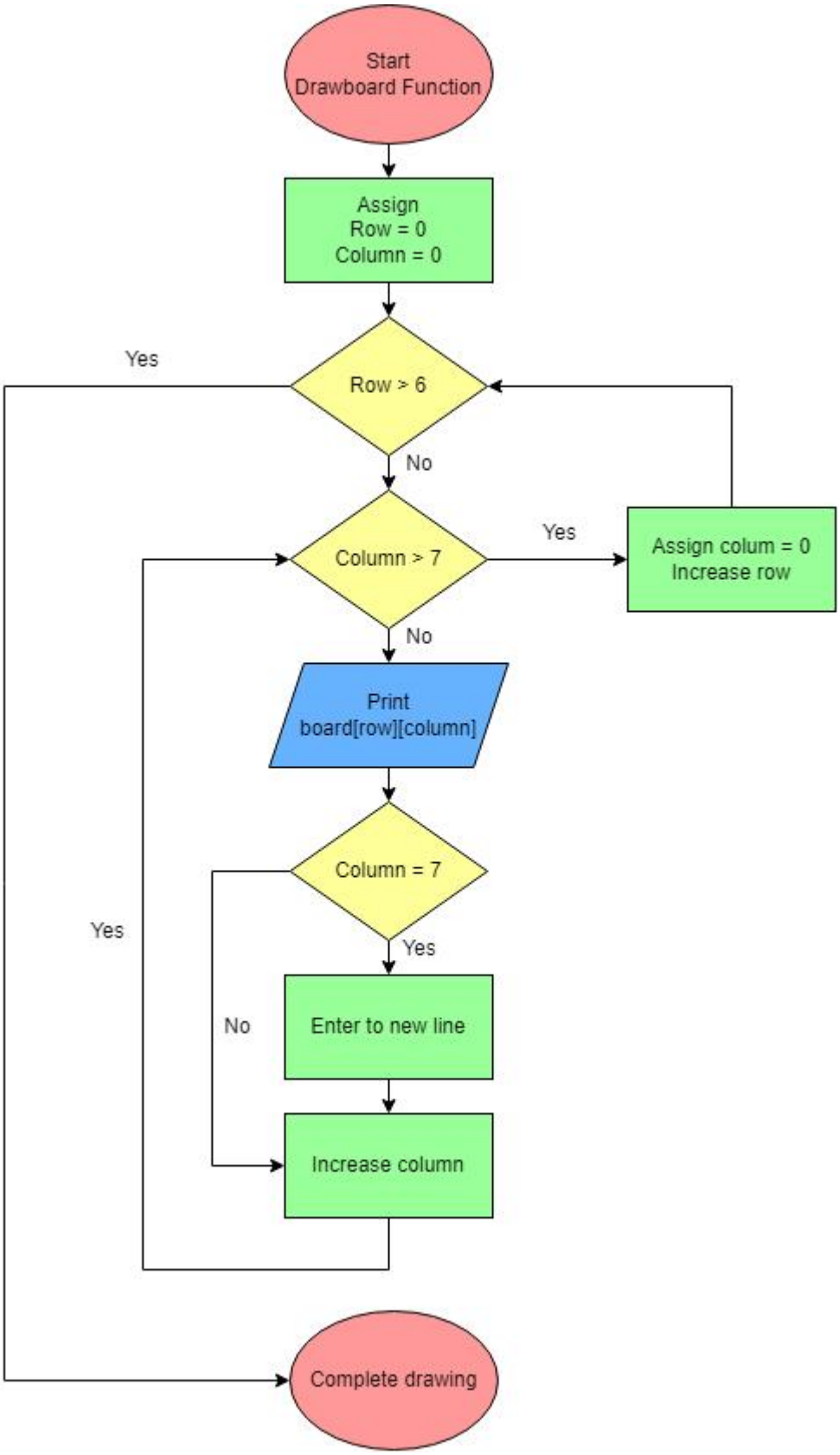
It has 13 rows and 29 columns but when there are any changes to the board (drop or remove pieces), it only works with position having row 1, 3, 5, 7, 9, 11 (6 rows) and column 2, 6, 10, 14, 18, 22, 26 (7 columns) because these positions are the place containing pieces.

2) Drawing board:

Explanation:

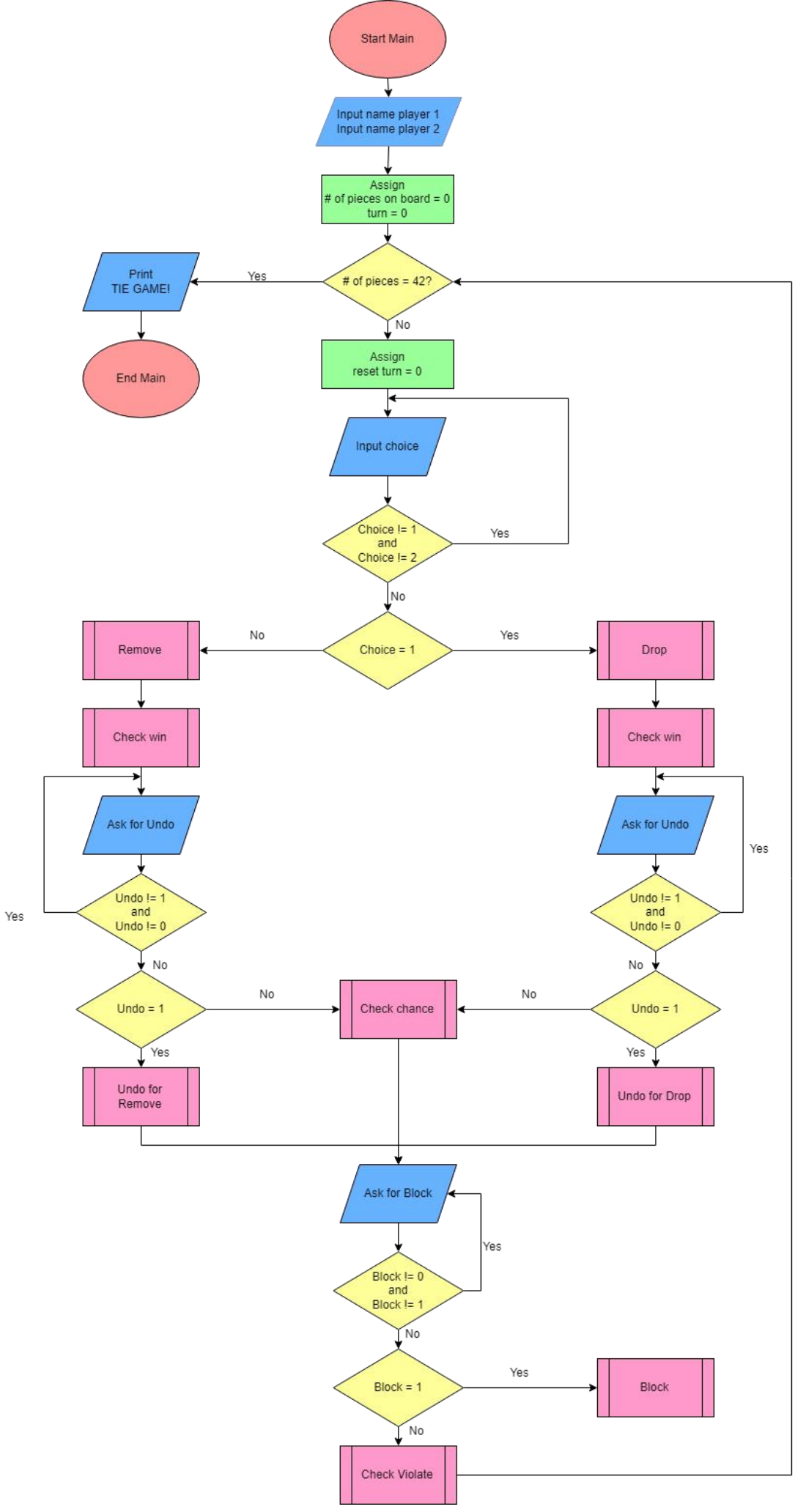
To print the board, I use 2 loops to traverse all elements of the board and print them to Run I/O. With the inner loop (loop for column), when it reaches the maximum column, the program print out the endline “\n” to enter to new line.

Flowchart:



3) Main function:

Flowchart:


Start Main
Input name player 1
Input name player 2
Assign
# of pieces on board = 0
turn = 0
# of pieces = 42?
Yes
Print
TIE GAME!
End Main
No
Assign
reset turn = 0
Input choice
Choice != 1
and
Choice != 2
Yes
No
Choice = 1
No
Remove
Yes
Drop
Check win
Check win
Ask for Undo
Ask for Undo
Yes
Undo != 1
and
Undo != 0
Yes
Undo != 1
and
Undo != 0
No
No
Undo = 1
No
Check chance
No
Undo = 1
Yes
Yes
Undo for
Remove
Undo for Drop
Ask for Block
Yes
Block != 0
and
Block != 1
No
Block = 1
Yes
Block
No
Check Violate

Explanation:

Firstly, I use 2 registers $s0 and $s1 to store value of the number of pieces on the board (play area) and turn (s1=0 => turn of first player, s1=1 => turn of second player), respectively. Then, I use a loop to process this game, it will loop until the board is full and no player wins.

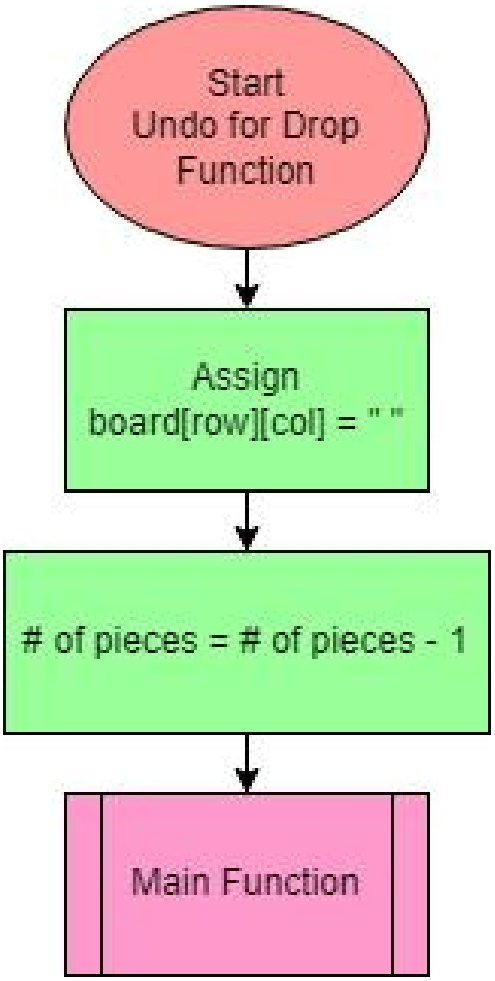
Secondly, in each turn, I'll ask player to choose between dropping a piece (enter 1) and removing an arbitrary piece of their opponent (enter 2). Therefore, there are 3 cases happening:

- **Case 1:** The player chooses a number different from 1 and 2: I ask player to enter again until they choose 1 or 2 (this is not counted as violation).
- **Case 2:** The player chooses 1 (drop a piece): I ask player to choose a column that they want to drop. In case they enter a number that is not in range 1-7 (column starts from 1 to 7), they will be counted 1 violation time and the turn $s1 is reset for them. In case they enter a valid number, 2 subroutines "drop" and "checkwin" will be called respectively (I will explain these 2 functions later).
- **Case 3:** The player chooses 2 (remove their opponent's piece): I ask player to choose a position containing their opponent's piece. Then, 2 subroutines "removepiece" and "checkwin" will be called (I will explain these 2 functions later). Note: If the chosen cell does not contain anything or it contains the piece of the player chose this function, it will be counted 1 time violation. Besides, the player's turn is also reset.

Thirdly, after dropping or removing a piece, every player will be asked to choose whether they want to undo their move (Enter 1 for Yes and 0 for No, if player enter another number, the program will ask again). In case the player chooses 0, the program jumps to "ask for block". Otherwise, it will process undo. There are also 2 cases for Undo (Undo of Remove or Undo of Drop):

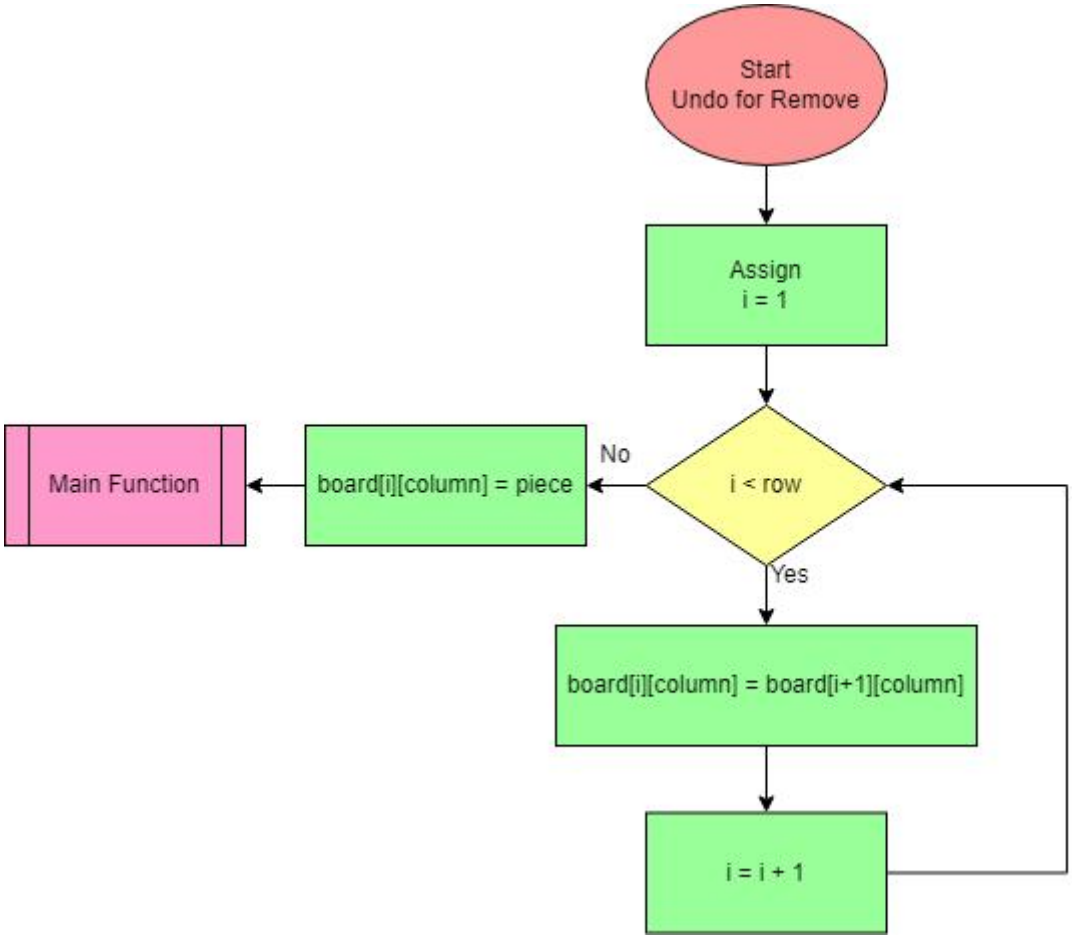
- **Undo of Drop:** I change the position that has been dropped the piece to space “ ” (meaning that delete the piece that has been dropped recently) and then decrease the number of pieces on board ($s0 = $s0 - 1).

Flowchart:



- **Undo for Remove:** I use a loop to traverse the elements in the column that the player chooses to remove until it reaches the chosen position and in each loop, the program copies data from the next element to the current element. Finally, I assign the piece that is removed to the chosen position.

Flowchart:



Next, after asking for undo, the program will ask player to decide if they want to block the opponent's next move. If the player chooses 0 (not block), the program jumps to check violation (I will explain later) and register $s1 (turn) will be changed to the opponent's turn (change to 1 if opponent is the 2nd player and 0 if opponent is the 1st player). In case, the player chooses to block their opponent's next move, their block time will be decrease (to 0), it means that they can not block the opponent one more time (if they still try to block one more time, it will be counted as 1 violation as mentioned above).

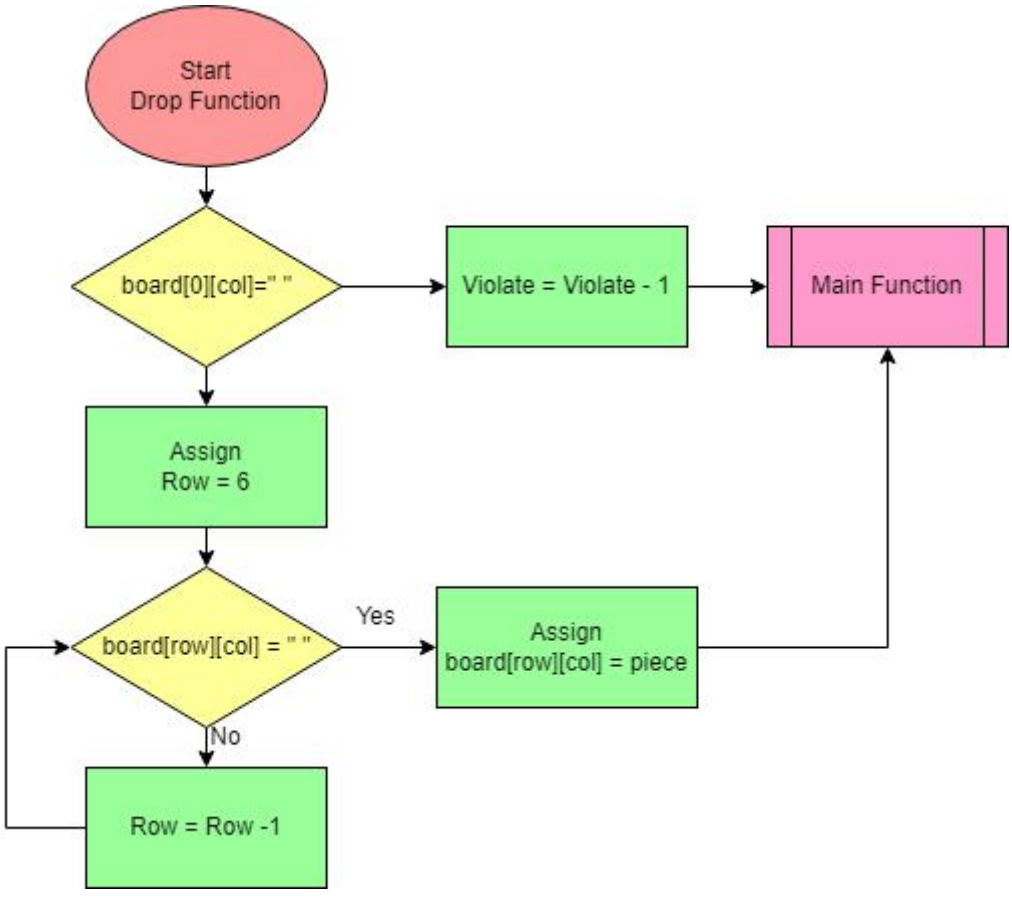
Last but not least, about checking violation that I mentioned in "block" above, after loading data of violation of player to a register, the program will check whether the value of that register is less than 0 or not (I assign 3 to violation and everytime the player violates, it will decrease by 1). In case, the program find out that the player violates more than 3 times, it will print out the winner (the opponent) immediately and stop game.

4) Drop Subroutine:

Explanation:

Firstly, I check that if the column is full or not. If it's full, the program will count 1 violation for the player and also reset their turn. If not, my program uses a loop to traverse in the column the player chooses to drop from bottom (row = 6) to top (row = 1) until it reaches a cell containing space " "(empty). Then it stores the piece to that position and increases the number of pieces on the board by 1.

Flowchart:

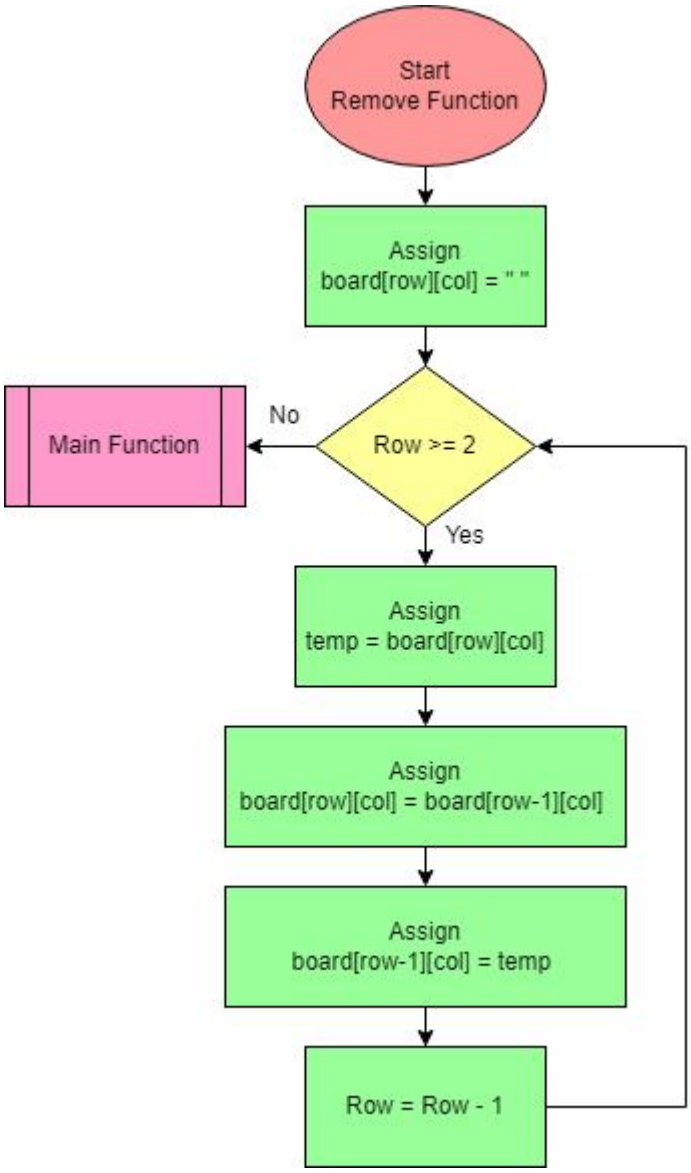


5) Remove Subroutine:

Explanation:

I store the space to the chosen position and then using loop from bottom to top (the column is chosen by player before) until it reaches the top row (row=1). In each loop, I swap value of the current cell and the next cell (above cell). The loop only process to row 2 (stop when reaches row 1) because in the last row (row = 1), we can not swap its value with row 0 (invalid row) so I just swap value of row 2 and row 1 then exit subroutine and jump.

Flowchart:



6) Check win:

The idea behind this function is that I'll check 4 times at a cell including check horizontal, check vertical, check right diagonal, check left diagonal:

a) Check horizontal:

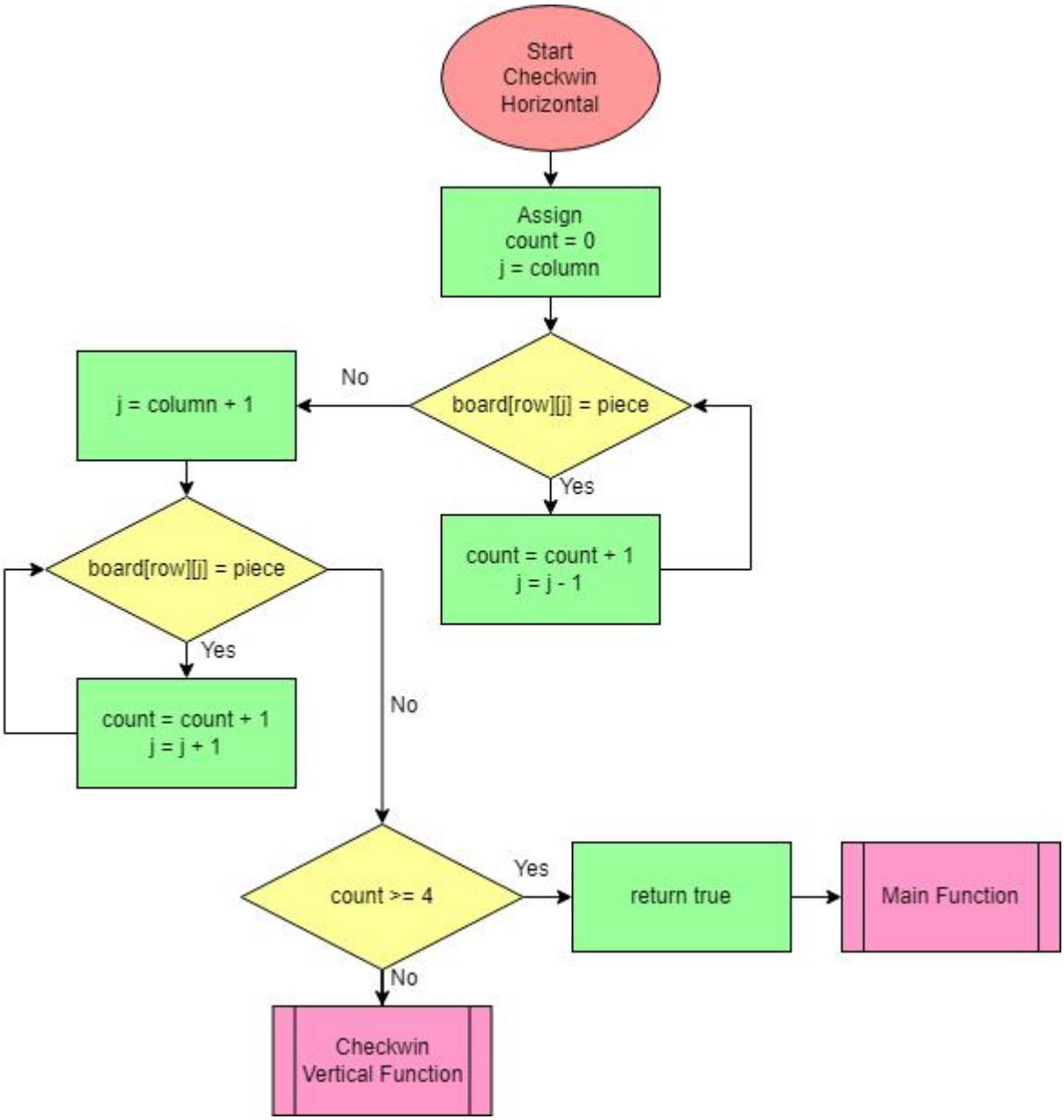
			O			
			O			
			O			
		X	X	**X**	X	

As you can see, the winner has 4 pieces "X" at the bottom of the board. For example, the recent drop is at row 6, column 5 (the red "X"), my program will check from that position to the left until it reaches a different piece ("O" or empty cell) and in with every cells containing "X", it will count up 1. After check horizontal (to the left), it will continue checking from the right of that position and loop the same as doing with the left.

Example: Using above case, the red "X" is the position. Here, I check from this position to the left and after finishing the program can find out 3 pieces "X" and count is update to 3 (position [6][5], [6][4], [6][3]). Then, I continue checking from the right of the red "X" (not from red "X" because it has been counted when check to left) and the program see that there is 1 "X" from the right of red "X" so it updates count to 4.

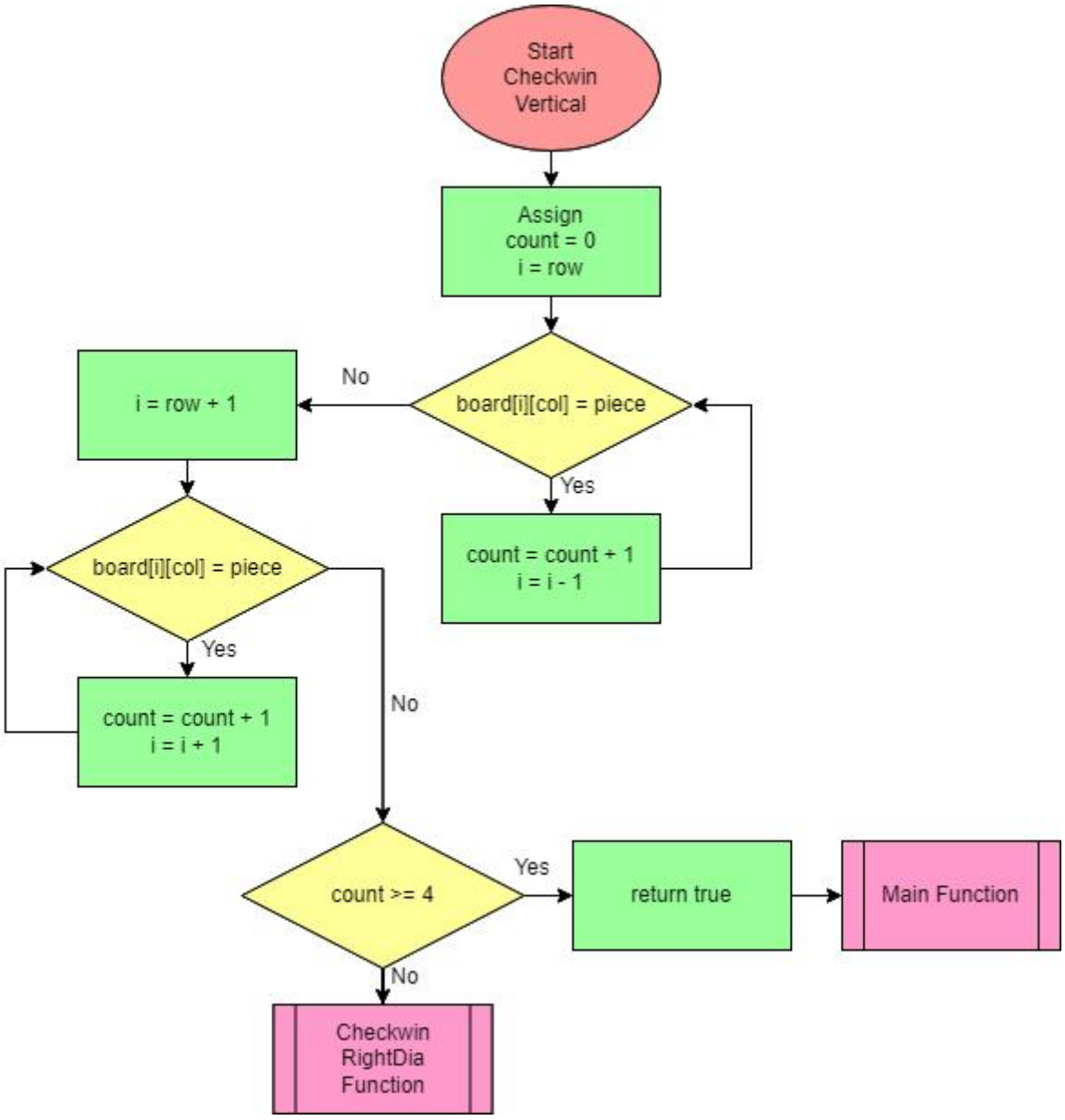
After processing checking horizontal, the program will check whether count is greater than or equal to 4. If yes, it will jump to main function and display

information of the winner to the screen. If no, it will reset the value of count to 0 and jump to check vertical. Here is the flowchart:



b) Check vertical:

The idea is also the same as horizontal, but I'll check from red "X" to top and then from above red "X" to bottom. If there's a winner (count >= 4), the subroutine jump to main and display information of the winner to the screen. If no, it will reset the value of count to 0 and jump to check right dia. Here is the flowchart:

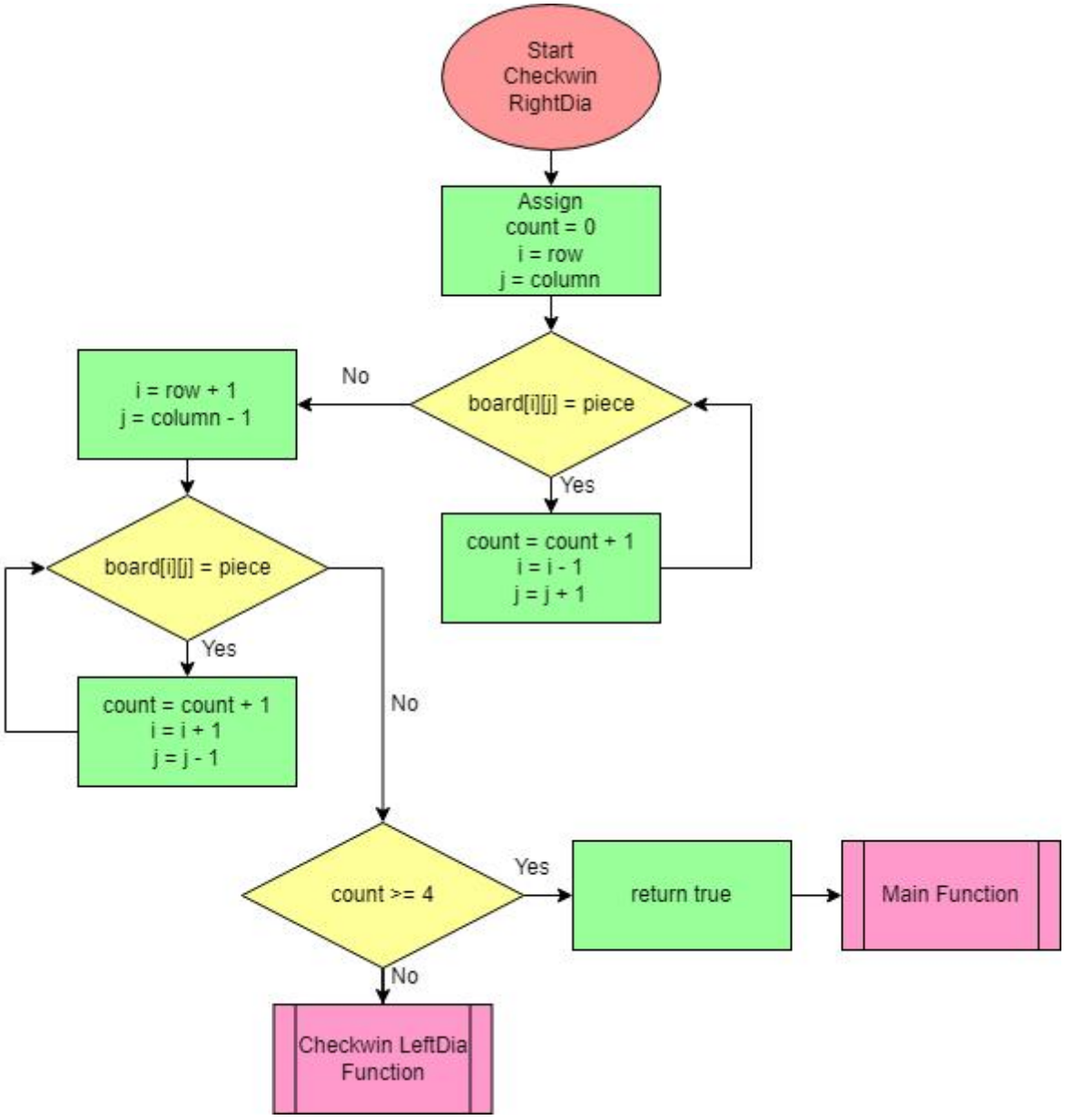


c) Check right dia:

						X
			X	O	**X**	X
			O	**X**	X	O
			X	O	O	O

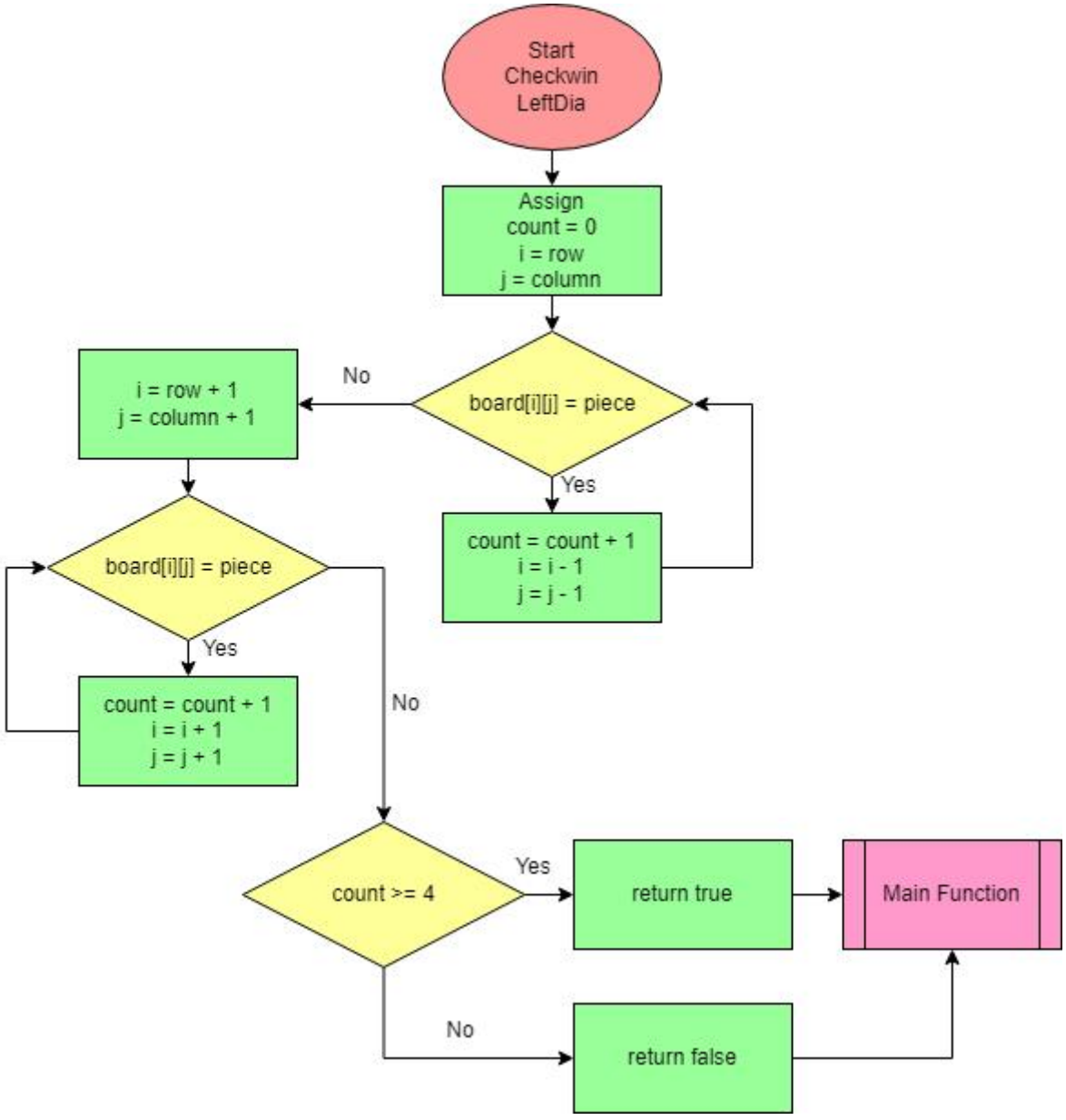
About diagonal, we have 2 cases, I mentioned right dia and left dia before and here is the example for right dia. The idea is also still the same as what I program with horizontal, but in diagonal when I check, I have to increase or decrease both column and row (in horizontal, I fix the row and only change column to check meanwhile in vertical, I fix the column and only change row to check). Here for example, I check from the underlined red "X" to right and up. It means that I will check from that position and in every loop, I increase column (because check to the right) and decrease row (check up). After finishing checking up, I check down (increase row, decrease column).

If count is greater than or equal to 4, the program jump to main function and announce the winner. If no, it continue resetting the value of count to 0 and checking left dia. Here is the flowchart:


Start
Checkwin
RightDia
Assign
count = 0
i = row
j = column
board[i][j] = piece
No
i = row + 1
j = column - 1
Yes
count = count + 1
i = i - 1
j = j + 1
board[i][j] = piece
Yes
count = count + 1
i = i + 1
j = j - 1
No
count >= 4
Yes
return true
Main Function
No
Checkwin LeftDia
Function

d) Check left dia:

The idea is the same as check right dia. Here is the flowchart:



If count is still not greater than or equal to 4 (it means that position does not make a 4 connecting pieces) the program will jump to main and continue

processing in main function. If count is greater than or equal to 4, the program jump to main and announce the winner.

NOTE: I will check win after dropping or removing:

- With drop, I check at the position that has been recently dropped.
- With remove, I check all cells from the position that is chosen to remove to the top piece.

7) Check chance to win:

I use the idea the same as check win but the condition is changed to whether count is greater than or equal to 3 or not.

IV) Conclusion:

After completing this assignment, I'm used to using MAR MIPS and this is also a great chance for me to improve my programming skill. I did learn a lot about storing, using, processing with data, how to use memory efficiently.

---------- THE END ----------